Programming Fundamentals- I

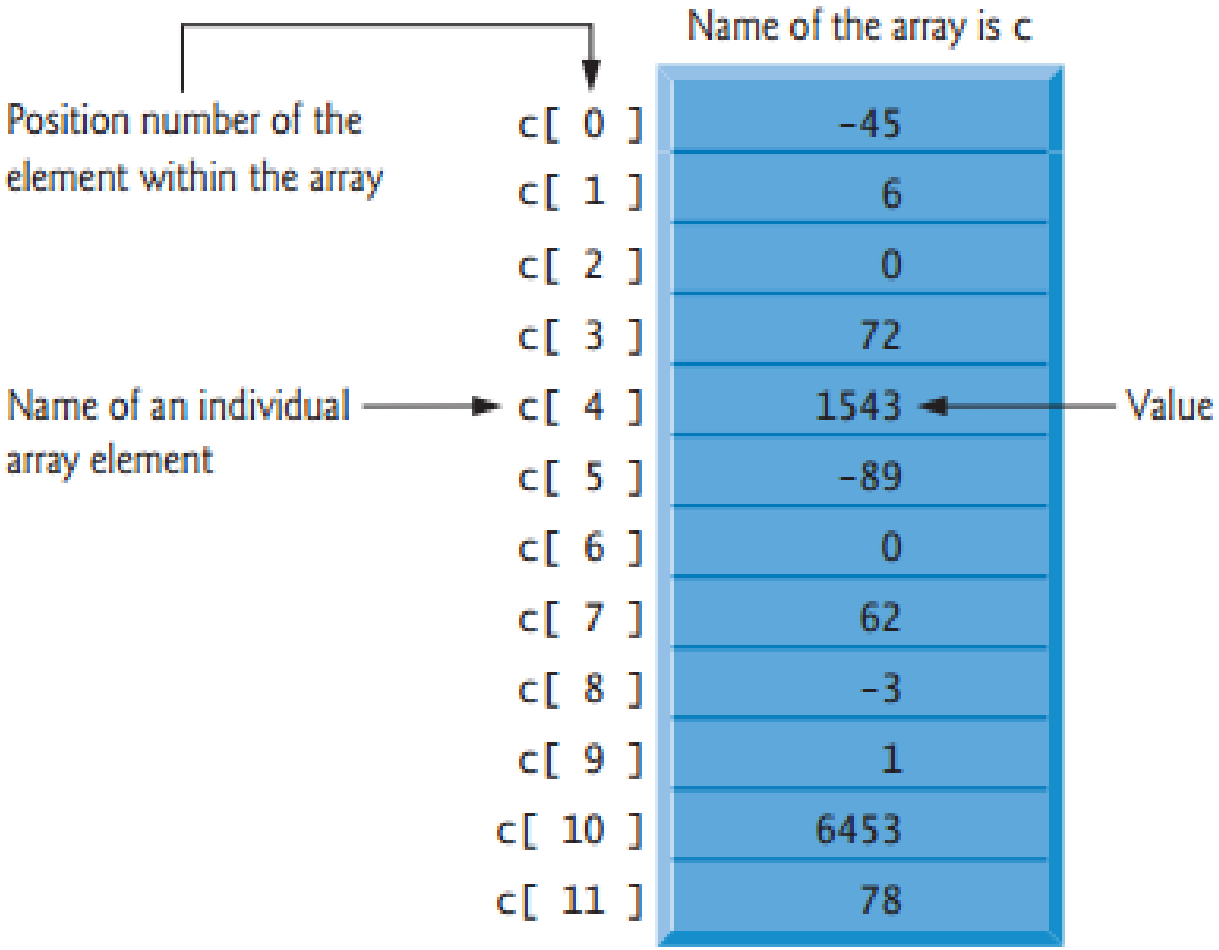
Rao Muhammad Umer
Lecturer,
Web: raoumer.github.io
Department of Computer Science & IT,
The University of Lahore.

Administrative Stuff

- Due Date for Assignment # 2 : PF – I Basic Concepts
  - **08/12/2016**

Arrays

Arrays


Name of the array is c
Position number of the element within the array
Name of an individual array element
Value
c[0]
c[1]
c[2]
c[3]
c[4]
c[5]
c[6]
c[7]
c[8]
c[9]
c[10]
c[11]
-45
6
0
72
1543
-89
0
62
-3
1
6453
78

Declaring an Array and Using a Loop to Initialize the Array's Elements

```
// Initializing an array's elements to zerosand printing the array.
#include <iostream>
using namespace std;
int main()
{
int n[ 10 ]; // n is an array of 10 integers

// initialize elements of array n to 0
for ( int i= 0;i < 10;++i )
    n[ i] = 0; // set element at location i to 0

// output each array element's value
for ( int j= 0;j < 10;++j )
    cout << n[j ]<<endl;
return 0;
}
```

Initializing an Array in a Declaration with an Initializer List

```
// Initializing an array's elements to zerosand printing the array.
#include <iostream>
using namespace std;

int main()
{
// use initializer list to initialize array n
int n[ 10 ]= { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
// output each array element's value
for ( int j= 0;j < 10;++j )
     cout << n[j ]<<endl;
return 0;
}
```

const

- **const** means a variable which doesn't vary –useful for physical constants or things like pi or e
  - You can also declare variables as being constants
  - Use the const qualifier:

```
const double pi=3.1415926;
const long double e = 2.718281828;
const int maxlength=2356;
const int val=(3*7+6)*5;
```

- (scientific) notation (mantissa/exponent)

```
const double PI = 3.14159e2;
```

- Sometimes const causes problems on some compilers.

#define

This preprocessor command replaces one thing with another before compilation

#define PI 3.14

#define GRAV_CONST 9.807

#define HELLO_WORLD "Hello World!\n"

NOTE – NO semi colon here!

Now, anywhere in the program we use PI or GRAV_CONST, it will be replaced with the replacement string BEFORE the real compiler starts (that's why we call it pre-processing)

```
c= 2.0 * PI * r;
a= GRAV_CONST * (m1*m2)/ (r * r);
```

Using Constants to Define Arrays

- It is useful to define arrays using constants:

```
#define MONTHS 12
int array [MONTHS];
```

Specifying an Array's Size with a Constant Variable and Setting Array Elements with Calculations

```
#include <iostream>
using namespace std;
int main()
{
// constant variable can be used to specify array size
const int arraySize = 10;
int s[ arraySize ]; // array s has 10 elements
for ( int i= 0;i <arraySize;++i ) // set the values
    s[ i] = 2 + 2 *i;
cout << "Element" << " \t"<< "Value" << endl;
// output contents of array s in tabular format
for ( int j= 0;j <arraySize; ++j )
  cout << j << "\t" <<s[j ]<<endl;
return 0;
}
```

Output

Element	Value
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20

2-D Arrays
Nesting in Loops

Multidimensional Arrays

- Arrays in C++ can have virtually as many dimensions as you want.
- Definition is accomplished by adding additional subscripts when it is defined.
- For example:
  - int a [4] [3] ;
  - defines a two dimensional array

    a[0][0] a[0][1] a[0][2]

    …

How to Print 2-D Arrays?

```
#include <iostream>
int main()
{
int a[4] [3] = { {1, 2, 3} , { 4, 5, 6} , {7, 8, 9} , {10, 11, 12} };
int row, col;
for (row = 0; row <=3; row++)
   { for (col= 0; col<=2; col++)
        {
             cout << a[row][col];
        }
    }
getch();
return 0;
}
```

Output of the Program

123456789101112

Output with Tabs

```
for (row = 0; row <=3; row++)
    { for (col= 0; col<=2; col++)
        {
           cout << a[row][col] << "\t";
         }
     }
```

Output:

1 2 3 4 5 6 7 8 9
10 11 12

Output in Matrix Form

```
for (row = 0; row <=3; row++)
    { for (col= 0; col<=2; col++)
        {
           cout << a[row][col] << "\t";
        }
       cout << endl;
     }
```

Output:

1	2	3
4	5	6
7	8	9
10	11	12

How to read (input) 2-D Arrays?

```
#include <iostream>
int main()
{
int a[4] [3];
int row, col;
for (row = 0; row <=3; row++)
{
   cout << "Enter 3 elements of row\n“ <<row + 1;
   for (col = 0; col <=2; col++)
      {
         cin >> a[row][col];
       }
 }
//Rest of the code goes here
```

Initializing Multidimensional Arrays

- The following initializes a[4][3]:

```
int a[4] [3] = { {1, 2, 3} , { 4, 5, 6} , {7, 8, 9} , {10, 11, 12} };
```

- Also can be done by:

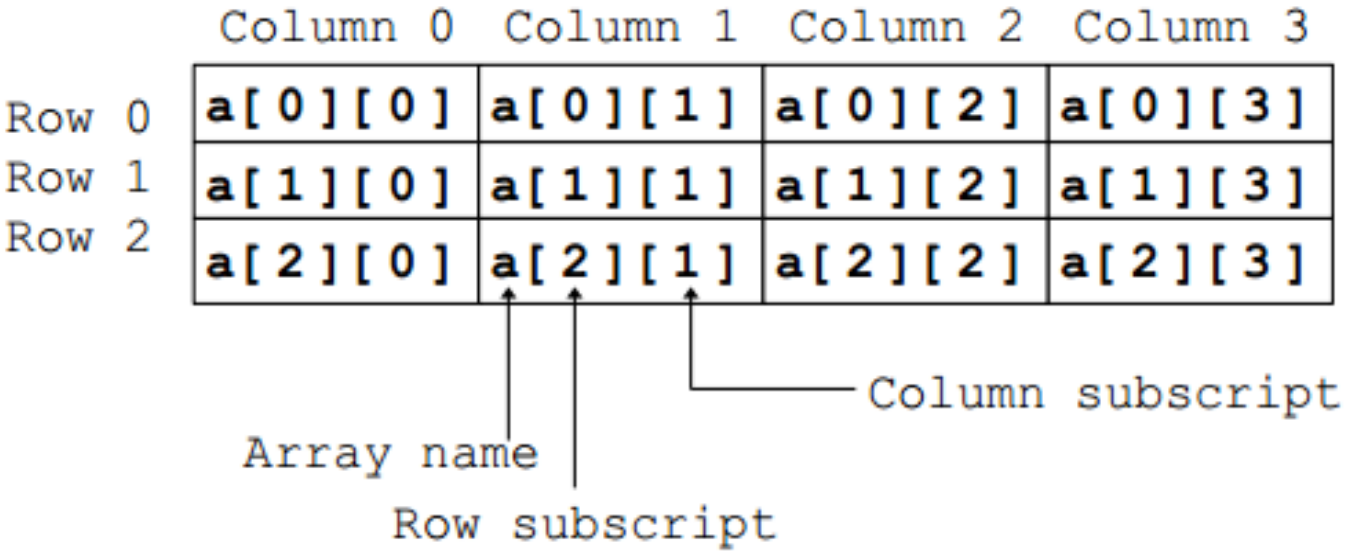
```
int a[4] [3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
```

  - is equivalent to

```
a[0][0] = 1;
a[0][1] = 2;
a[0][2] = 3;
a[1][0] = 4;
...
a[3][2] = 12;
```

Multiple-Subscripted Arrays

- Multiple subscripted arrays
  - Tables with rows and columns (**m** by **n** array)
  - Like matrices: specify row, then column



Multiple-Subscripted Arrays

- Initialization

1	2
3	4

  - `int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };`
  - Initializers grouped by row in braces
  - If not enough, unspecified elements set to zero

1	0
3	4

```
int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };
```

- Referencing elements
  - Specify row, then column

Addition of Two Matrices

```
#include <iostream>
using namespace std;
int main()
{
int X[2][2] = { {1,2},{3,4} },
Y[2][2] = { {5,6},{7,8} };
int add[2][2];
int i, j;
cout << "\n\tAdditionof two
matrices is";
for(i = 0; i<2; i++)
{
for(j = 0; j<2; j++)
{
add[i][j] = X[i][j] + Y[i][j];
cout <<"\t“<< add[i][j];
}
}
getch();
return 0;
}
```

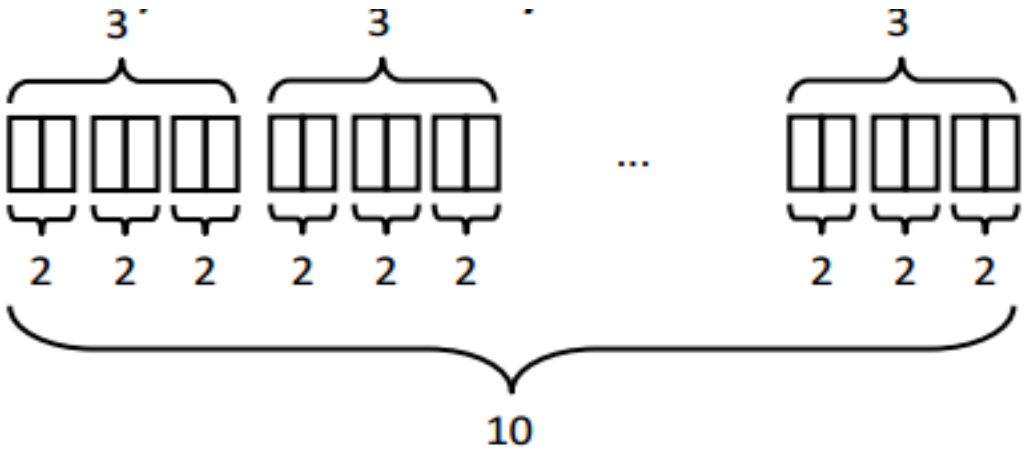
Multiplication of Two Matrices

```
#include <iostream>
using namespace std;
int main()
{
int X[2][2] = { {1,2},{3,4} }, Y[2][2] = {
{5,6},{7,8} };
int add[2][2], mul[2][2];
int i, j, k, sum = 0;
cout << "\n\n\t Multiplications of
two matrices is";
```

```
for (i= 0; i<2; i++)
 {
     cout << "\n\n\t";
     for (j = 0; j<2; j++)
       {
           for (k=0; k<2; k++)
            {
                 sum = sum + (X[i][k]*Y[k][j]);
            }
           mul[i][j] = sum;
           cout << "\t“ <<  mul[i][j];
           sum = 0;
       }
  }
getch();    return 0;   }
```

Multidimensional Arrays

- Array declarations read right-to-left
- int a[10][3][2];
- “an array of ten arrays of three arrays of two (type ints)”. In memory



Infinite loops

Infinite loops

```
// Infinite Loops
#include<iostream>
int main()
{
int i= 7;
while (1) /* any non zero number can be provided here. It is an
            infinite loop */
    {
            cout << "\n“ << i;
             i++;
    }
getch();
return 0;
}
```

```
// Infinite loop using while loop and break statement
#include<iostream>
int main()
{
int i;
cout << "Please enter an integer number from 10 to 20\n";
while (1)
    {
        cin >> i;
        if (i>= 10 && i<=20)
                break;
        else
                cout << "Invalid. Please enter an integer number from 10
                 to 20\n");
     }
// Rest of the code goes here
```

```
// Infinite loop using for loop and break statement
#include<iostream>
int main()
{
int i;
cout << "Please enter an integer number from 10 to 20\n";
for ( ; ; )
    {
        cin >> i;
        if (i>= 10 && i<=20)
            break;
        else
            cout << "Invalid. Please enter an integer number from 10
             to 20\n");
     }
// Rest of the code goes here
```

```
// Infinite loop using do-while loop and break statement
#include<iostream>
int main()
{
int i;
cout << "Please enter an integer number from 10 to 20\n";
do
    {
        cin >> i;
        if (i>= 10 && i<=20)
                break;
        else
                cout << "Invalid. Please enter an integer number from 10
                 to 20\n");
     } while (1);
// Rest of the code goes here
```

break, continue statements

The break statement

```
//More about break
/* if used in loop, it only breaks the
loop and then executes next
statements */
#include<iostream>
int main()
{
int i, n, prime = 1;
cout << "Enter a positive integer";
cin >> n;
if (n ==2)
    cout << "The number %d is a
            prime number\n", n);
else
    {
for (i= 2; i<=n/2; i++)
            { if(n%i== 0)
                  { prime = 0;
                    break;
                   }
  }
if (prime == 0)
   cout << "The number" << n << "is
not a prime
            number\n";
else
    cout << "The number" << n << " is
            a prime number\n";
}
getch();return 0; }
```

The break Statement

- **break**
  - Causes immediate exit from a while, for, do/while or switch structure
  - Program execution continues with the first statement after the structure
  - Common uses of the break statement
    - Escape early from a loop
    - Skip the remainder of a switch structure

Example: for and break Together

```
int mynum= 3;
int guess;
for(;;)
{
   cout << "Guess my number:");
   cin >>guess;
   if (guess==mynum)
     {
          cout << "Good guess!\n";
          break;
     }
   else
          cout << "Try again.\n";
}
```

The notation for(;;) is used to create an infinite for loop. while(1) creates an infinite while loop instead.

To get out of an infinite loop like this one, we have to use the break statement.

The continue statement

```
/*Program to determine how many
vowels are in a line of text*/
#include<iostream>
#include<conio.h>
int main()
{
char sentence [50];
int i, nvowels= 0, nconsonants= 0;
cout << "Enter a line of text");
cin >> sentence;
for (i= 0; sentence[i] !='\0'; i++)
  {
      if ( sentence[i] == 'a‘
      || sentence[i] == 'e'
      || sentence[i] == 'i'
      || sentence[i] == 'o‘
      || sentence[i] == 'u‘
      || sentence[i] == 'A
      || sentence[i] == 'E'
      || sentence[i] == 'I'
      || sentence[i] == 'O'
      || sentence[i] == 'U'  )
      {
            nvowels++;
            continue;
      }
 if (sentence[i] != ' ')
      nconsonants++;
    }
cout << "No. of Vowels are“ << nvowels <<
endl;
cout << "No. of Consonants are\n“ <<
nconsonants;
getch();
return 0;
}
```

The continue statement

Used for skipping the remainder of the body of a **while, for or do/while** structure and proceeding with the next iteration of the loop

- **while and do/while**
  - Loop-continuation test is evaluated immediately after the **continue** statement is executed
- **for**
  - Increment expression is executed, then the loop-continuation test is evaluated

Some Jargons

- **Loop**
  - Group of instructions computer executes repeatedly while some condition remains true
- **Counter-controlled repetition**
  - Definite repetition: know how many times loop will execute for (i= 0; i< 12; i++)
  - Control variable (e.g i)used to count repetitions
- **Sentinel-controlled repetition**
  - Indefinite repetition
  - Used when number of repetitions not known
  - Sentinel value indicates "end of data"

Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires
  - The name of a control variable (or loop counter)
  - The initial value of the control variable
  - A condition that tests for the final value of the control variable (i.e., whether looping should continue)
  - An increment (or decrement) by which the control variable is modified each time through the loop

Essentials of Counter-Controlled Repetition

- Example:

```
int counter = 1; // initialization
while ( counter <= 10 ) { // repetition condition
cout << counter << endl ;
++counter; // increment
}
```

  - The statement

    int counter = 1;

    - Names **counter**
    - Declares it to be an integer
    - Reserves space for it in memory
    - Sets it to an initial value of 1